UNIVERSIDAD DE ZARAGOZA

Proyecto Fin de Carrera

Ingeniería Informática

ENTORNO INMERSIVO DE BAJO COSTE CON INTERACCIÓN NATURAL

Grupo de Informática Gráfica Avanzada

Departamento de Informática e Ingeniería de Sistemas

Centro Politécnico Superior

Curso académico: 2010-2011

Fecha: Abril de 2011

Autor: Arturo Alastuey Elpuente

Director: Jorge López Moreno

Ponente: Francisco José Serón Arbeloa

Entorno inmersivo de bajo coste con interacción natural

Resumen

El presente proyecto consiste en el desarrollo de una aplicación inmersiva de bajo coste que dispone de interacción natural con el usuario.

Esta aplicación dispone de características como la visualización en tres dimensiones, el reconocimiento de la posición espacial del usuario (*Head Tracking*), el soporte para utilizar técnicas de render avanzado o la posibilidad de desplazarse por la escena mediante un sistema de navegación.

Para conseguir la visualización en tres dimensiones, dispone de un sistema de estereoscopía que renderiza dos imágenes similares de la escena, cada una de ellas con la perspectiva ligeramente cambiada. Cada ojo del usuario debe visualizar solamente una de estas dos imágenes, simulando así la perspectiva 3D del mundo real. Para comprobar el funcionamiento se ha utilizando un sistema basado en la polarización en el cual el usuario debe ponerse unas gafas que le permiten visualizar una imagen en cada ojo. A la hora de determinar los puntos de enfoque de estas dos cámaras, se ha tenido en cuenta que no afecten a la percepción del escalado por parte del usuario.

Para la realización de un correcto *Head Tracking*, es necesario posicionar la cabeza del usuario y modificar la perspectiva de la escena respondiendo a los movimientos que realice. Esto provoca una gran sensación de inmersión en la escena. Para conseguir el posicionamiento se han utilizado los mandos de la consola *Wii* de *Nintendo*, debido a su bajo coste y a la buena relación del rendimiento con respecto al mismo.

El sistema de visualización de la aplicación permite hacer uso de técnicas avanzadas de render. Una de las técnicas que pueden utilizarse son los *shaders* (pequeños programas que calculan el aspecto final o color de cada píxel a pintar en pantalla). Como demostración se ha creado un *shader* que implementa la novedosa técnica de *parallax mapping*. Esta técnica consigue mayor sensación de profundidad de las texturas con menor número de polígonos, reduciendo los tiempos y posibilitando el cálculo de la escena en tiempo real.

Algunos de los campos en los que puede aplicarse el trabajo realizado son la arqueología virtual, la biomedicina, la realidad aumentada y realidad virtual.

Agradecimientos

A mis padres, hermanos y mi abuela, por alentarme, darme fuerzas y por brindarme todo su apoyo y ayuda durante todo este tiempo, a Elisa porque siempre ha estado ahí cuando lo he necesitado, al resto de amigos y familia por darme su apoyo y confianza. Y a Jorge y Paco por ayudarme en todo lo posible y hacer mucho más de lo que marcan sus funciones.

Índice General

Capítulo 1. Introducción

1.1. Estructura de la memoria

La memoria consta de los siguientes capítulos:

1. **Introducción**.
   Ofrece una visión general del proyecto y de su documentación
2. **Mundo Virtual.**
   Describe las principales características del mundo virtual de la aplicación.
3. **Generación de estereoscopía.**
   Define y describe el concepto de estereoscopía y analiza las principales características de ésta que se han utilizado para el desarrollo de este proyecto.
4. **Interacción**
   Describe las diferentes formas de interacción de la aplicación con el usuario.
5. **Descripción de la aplicación.**
   Describe la arquitectura, partes físicas y funcionamiento de la aplicación.
6. **Resultados y conclusiones.**
   Realiza una valoración del trabajo realizado. Además, analiza el cumplimiento de los objetivos planteados y las posibilidades de continuación, así como las posibles aplicaciones. También explica los problemas encontrados durante el desarrollo de proyecto.

1.2. Ámbito y contexto

Este proyecto fin de carrera ha sido desarrollado como parte de la labor investigadora del *Grupo de Informática Gráfica Avanzada* (*GIGA*), que pertenece al *Departamento de Informática e Ingeniería de Sistemas* (*DIIS*) de la *Universidad de Zaragoza*. Más concretamente, forma parte del importante proyecto de investigación de ámbito estatal financiado por el Ministerio de Ciencia y Tecnología *"TANGIBLE: Humanos Virtuales realistas e interacción natural y tangible"*, en cuya descripción se expone como uno de sus objetivos: "La interacción con el usuario, utilizando el paradigma de los interfaces naturales y multimodales, incluidos los tangibles. Las aplicaciones más inmediatas de sus resultados se enmarcan en los entornos de la formación y el entretenimiento, aunque prevemos un campo de aplicación más extenso a medio y largo plazo."

Fue concedido en la convocatoria de *Programas Nacionales del Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica 2004-2007* del *Ministerio de Educación y Ciencia*. En el proyecto participan investigadores de la *Universidad de Zaragoza*, de la *Universidad de las Islas Baleares* y de la *Universidad Pública de Navarra*. El proyecto une tres grandes campos (informática gráfica, personajes 3D inteligentes e interfaces naturales) en una propuesta coherente cuyo

objetivo global es la generación de humanos virtuales realistas, tanto desde el punto de vista visual como de comportamiento, que soporten la interacción natural y emocional con el usuario. El presente proyecto se enmarca en el área que explora la inmersividad, y realismo con entornos virtuales. Se pretende ofrecer una solución de bajo coste capaz de proporcionar la capacidad de aumentar la sensación de inmersividad con la detección de la localización espacial del usuario para la inmersividad y la utilización de estereoscopia creando el efecto de la tercera dimensión. Al mismo tiempo que utiliza técnicas de *render* avanzadas para proporcionar una sensación mayor de realismo.

1.3. Medios

Los *Wiimotes* (abreviatura de *Wii remote*, mando de la Consola *Wii*) (Figura 1.1) se comunican con la consola *Wii* a través del protocolo de comunicación inalámbrica *Bluetooth*. A pesar de que *Nintendo* no ha liberado el significado de los paquetes de datos que se intercambian el mando y la consola, varios programadores han capturado y deducido el significado de gran parte de la información transmitida. A partir de este proceso de ingeniería inversa se han desarrollado (y siguen en constante actualización) varias bibliotecas sobre diferentes entornos y plataformas que proporcionan un cómodo análisis de la información. La biblioteca que motivó este proyecto fue *WiimoteLib*, de Brian Peek. El nacimiento de las primeras versiones de estas bibliotecas, en primavera de 2007, creó una atractiva línea de exploración de las posibilidades de interacción que proporciona un periférico tan versátil como éste. En la actualidad hay multitud de proyectos en curso y supone un campo en continua evolución.

La característica del control *Wiimote* que se explota en este proyecto es su capacidad de captar puntos infrarrojos: en su cara frontal se encuentra una sencilla cámara que capta hasta cuatro focos de luz que sirven para deducir su orientación respecto a la mal llamada *Sensor bar*, ya que no tiene sensores, sino emisores de luz infrarroja en las zonas oscuras de sus extremos frontales.

Figura 1.1 Wiimote y Sensor bar

1.4. Motivación y objetivos

La idea de la base del proyecto surgió de un trabajo de Johnny Chung Lee [1], que tuvo cierta relevancia en Internet, en el que se demostraba la detección de los

reflejos de luz infrarroja por parte de las cámaras de los mandos de la consola *Wii* de *Nintendo.*

Uno de sus proyectos es el llamado *Head Tracking for Desktop VR Displays using the Wii Remote* (Localización de la cabeza para visualizadores domésticos de realidad virtual con el mando de la *Wii*). El proyecto exploraba ciertas capacidades de localización espacial, que podían ser ampliadas cambiando ligeramente el enfoque: Para la localización se utilizaba un solo mando y necesitaba dos emisores infrarrojos (Figura 1.2).

Figura 1.2 Johnny Chung Lee demuestra el funcionamiento de su "Head tracker"

Basándose en sus posiciones y su distancia relativa, estimaba la distancia entre los emisores y los mandos. Esto divide por dos el máximo de cuatro puntos detectados por el sistema; limitación que se puede evitar utilizando dos cámaras en vez de una. Este proyecto se basa en estos mismos conceptos.

La aplicación imaginada por los directores del proyecto era la consecución de una interfaz inmersiva completa y de bajo coste compuesta por un sistema de localización espacial para la ubicación automática del punto de vista del usuario, así como de los cursores destinados a la interacción. Además incluiría un sistema de reconocimiento de gestos básicos. La localización del punto de vista del usuario se utilizaría para actualizar la imagen congruentemente con la posición de su cabeza y crear mayor realismo. Como primera aplicación, se pensó en utilizarlo para navegar sobre entornos 3D virtuales creados en el propio *GIGA*. Este objetivo, por su amplitud y gran complejidad, fue dividido en varios proyectos para desarrollar en paralelo.

El presente proyecto es el encargado de la creación del mundo virtual en 3D utilizando técnicas de render avanzado para mayor realismo, la localización espacial del usuario para crear una sensación de inmersividad mayor y la capacidad de cargar en el

mundo virtual los entornos 3D virtuales creados en el GIGA. Todo esto siendo lo suficientemente modular, flexible, completo y claro como para ser utilizado por los demás proyectos involucrados.

1.5. Alcance

1.5.1. Resultado

La aplicación resultante de este proyecto fin de carrera proporciona las herramientas necesarias para que cualquier usuario pueda cargar un modelo 3D y visualizarlo en un entorno inmersivo. Sus características más importantes son la localización espacial del usuario y la estereoscopia, que sirve para aumentar la inmersividad. Además, utiliza técnicas avanzadas de *renderizado* que permiten aumentar el realismo de la aplicación.

1.5.2. Esfuerzo

Para el desarrollo de esta aplicación ha sido necesario el estudio de algunas herramientas y librerías externas, así como la utilización de las mismas. Algunas de las más importantes han sido *OGRE (Open Source 3D Graphics Engine)* [2], diferentes exportadores de escenas de *3D Studio* e importadores en la aplicación y la librería *WiimoteLib* para la localización espacial del usuario. Además, ha sido necesario el estudio e implementación de gran cantidad de conceptos de estereoscopía, así como de técnicas de *render* avanzado utilizando *shaders*, para su posterior integración en la aplicación.

1.5.3. Futuro

Este proyecto establece un sistema de visualización tridimensional que podrá utilizarse para diferentes propósitos en ámbitos tan distintos como la arqueología virtual, la interacción con avatares o la biomedicina. Además, podrá integrarse con distintos módulos de interacción.

1.6. Tareas del proyecto

Una vez estuvieron fijados los objetivos del proyecto, se dividió el trabajo a realizar en diferentes partes. Como resultado de esta planificación se obtuvieron las siguientes tareas a realizar:

- Estudio de las bases teóricas necesarias, aprendizaje y familiarización con el entorno y el lenguaje de programación.

Para poder abordar la consecución de los objetivos con los conocimientos apropiados, ha sido necesario el estudio de algunas áreas de la geometría, especialmente las relacionadas con la visión estereoscópica. Los conceptos aprendidos han sido utilizados para calcular la posición espacial de las dos cámaras que conforman la cámara estereoscópica, simulando la visión desde los ojos de un ser humano y provocando así el efecto de las tres dimensiones.

También ha sido necesario el estudio de diferentes lenguajes de programación y de algunas de las bibliotecas disponibles. El lenguaje de programación utilizado ha sido *C#* [13] *y* como entorno de programación se ha utilizado el potente entorno de desarrollo *Microsoft Visual Studio* [3].

- Comprensión, análisis, selección e integración de los diferentes recursos.

La utilización de un entorno de programación mayoritario como *Microsoft Visual C# Express Studio* permite la integración de recursos externos que simplifican la tarea de codificación del sistema. Se estudiaron las diferentes herramientas disponibles, buscando las que proporcionaran funcionalidades de utilidad en este proyecto, evitando así la implementación de algo que ya se hubiera desarrollado antes. Para la interacción con los *Wiimotes* se ha utilizado y modificado la biblioteca *WiimoteLib*. Por otro lado, para exportar escenas a un archivo desde un programa comercial de desarrollo 3D (como por ejemplo 3D Studio [19]) se ha utilizado el exportador *ofusion* [4]. Para importar estos archivos al mundo virtual creado en la aplicación desarrollada en este proyecto se ha utilizado la librería *osmLoader 2* [5].

- Configuración del sistema

Para el correcto funcionamiento de la aplicación ha sido necesaria la configuración de los diferentes elementos que se integran en la aplicación (*ogre, osmloader 2, wiimote, shaders*) para que el funcionamiento global del sistema sea el adecuado.

- Desarrollo del sistema: implementación y pruebas.

El modelo de proceso elegido para el desarrollo de este proyecto es el modelo de cascada incremental (Figura 1.3). Esto significa que para llegar a tener resuelto el objetivo final se han ido resolviendo objetivos parciales intermedios. Cada uno de estos objetivos intermedios suponía una funcionalidad añadida, que era comprobada y revisada antes de continuar con la siguiente. Esto se hizo así debido a que algunas de las funcionalidades requerían del correcto funcionamiento de las anteriores para ser implementadas. Además, de esta manera se puede disponer antes del software, aunque no sea de manera completa.

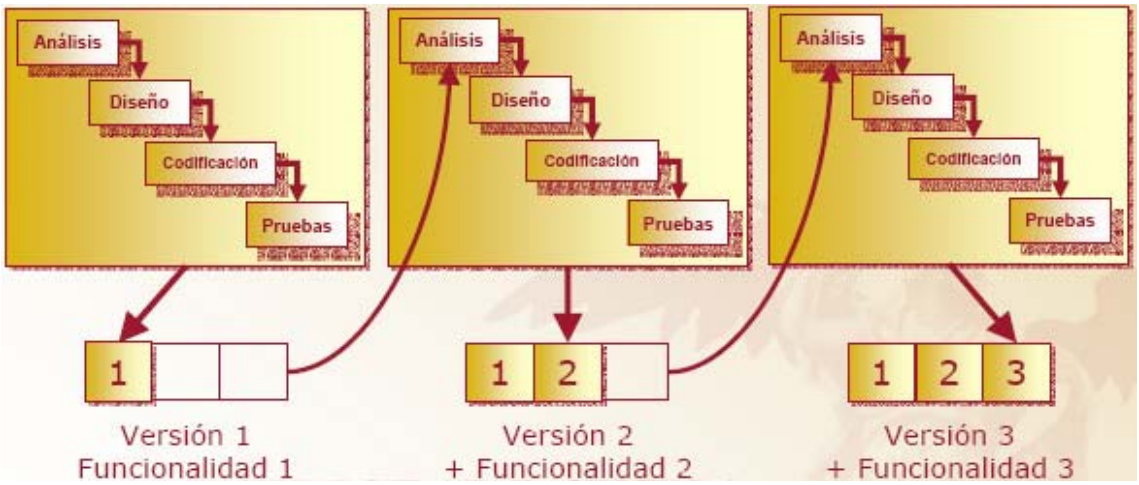

Figura 1.3 Modelo de proceso en cascada incremental

Los objetivos intermedios en los que se dividió el desarrollo de la aplicación fueron: creación de una escena simple importada desde un programa comercial de desarrollo 3D, posicionamiento del usuario utilizando la librería *WiimoteLib*, creación del sistema de estereoscopia e integración con la aplicación y por último integración de un sistema de *render* avanzado para una mayor calidad gráfica.

Capítulo 2. Mundo Virtual

2.1. Carga de objetos

Para poder formar la escena era necesario cargar los objetos en la aplicación. Para esto se sigue el siguiente proceso. Primero se exportaba la escena de un programa comercial de diseño 3D como el 3D Studio, y en segundo lugar se importaba esta escena en nuestra aplicación.

Para poder realizar este proceso había que elegir un formato de fichero, de tal forma que se pudiera exportar la escena desde el programa de diseño y posteriormente importarla en la aplicación. Se analizaron diferentes herramientas capaces de exportar del programa de diseño la escena e importarla en la aplicación.

En primer lugar se estudió *Ofusion*, éste es un exportador que se puede instalar en *3D Studio* y que exporta las escenas a un archivo con el formato .osm. En este archivo se guardan las referencias y posiciones a los distintos objetos que conforman la escena, dichos objetos se exportan al mismo tiempo a archivos con el formato .mesh. También se generan otros tipos de archivos para guardar otros datos, por ejemplo los materiales (.material).

Para cargar estas escenas en la aplicación se estudió el importador llamado *OSMSceneLoader 2* que es la versión mejorada de *OSMSceneLoader*, éste lee este tipo de archivos y los traduce al lenguaje de *Ogre* que es el que utiliza la aplicación.

En segundo lugar se estudió *OgreMax* [6] que al igual que el anterior es un exportador que se puede instalar en *3D Studio* y que exporta las escenas a un archivo con el formato .scene. En este se hace referencias a los objetos que están exportados a archivos con el formato .mesh.

Después de analizar estos dos exportadores se decidió utilizar el primero ya que se descubrió durante el proceso de análisis que la segunda opción estaba poco depurada y contenía diversos errores que hacían que el proceso no fuera lo suficientemente robusto. Con este exportador se pueden elegir diferentes opciones como, por ejemplo, qué elementos exportar (objetos, luces, cámaras, materiales, etc.) o qué propiedades tendrán los objetos exportados (las normales, la configuración de los ejes, etc.).

Para poder cargar un objeto en la aplicación debemos poner el archivo .osm junto con todos los archivos resultantes del proceso de exportación (.mesh, .material, imágenes, etc.) en una carpeta que sea visible desde la aplicación. En el archivo .xml de configuración de la aplicación es necesario añadir la ruta del archivo para que se cargue la escena correspondiente.

2.2. Materiales

Los objetos cargados en la aplicación debían de ser texturizables para poder estar dotados de una apariencia más realista, ya que los materiales procedentes del exportador muchas veces no son del todo realistas. También era necesario poder modificar las texturas de la escena en función de las necesidades.

Para que los materiales que texturizan un objeto fueran modificables se dotó a la aplicación de un sistema que permite cambiar la textura de un objeto de la escena de forma rápida e independiente de los demás objetos. Para esto se dispone de un archivo .material en el que están definidos todos los materiales de los objetos de la escena, en este archivo que provee *Ogre* se pueden definir materiales normales o materiales que utilicen técnicas como *shaders*.

Un *shader* básicamente es un pequeño programa que calcula el color del píxel a pintar en la pantalla, por cada píxel de la pantalla o por cada vértice de la figura, según el tipo de *shader*, se realiza una ejecución de este programa. Hay dos tipos de *shaders*, el *Vertex shader* que es el que realiza una ejecución por cada vértice de la figura, y el *Píxel shader* que es el que realiza una ejecución por cada píxel a pintar. Todos estos programas se ejecutan en la GPU (Graphics Processing Unit) y utilizan la API de *DirectX* [7] u *OpenGL* [8].

Se investigaron los distintos lenguajes de *shaders* disponibles en el mercado, los más usados son, *HLSL* que es un lenguaje propiedad de Microsoft que se debe utilizar con *DirectX*, *GLSL* que es un lenguaje del grupo *Khronos* y que está diseñado específicamente para su uso dentro del entorno de *OpenGL* y por último *CG* que es un lenguaje propiedad de *Nvidia* cuya principal ventaja es que puede ser usado con las Apis de *DirectX* y *OpenGL* aunque es recomendable crear programas específicos para las diferentes tarjetas.

Uno de los requisitos de nuestro sistema es que sea integrable a nivel gráfico con el software ya existente en el proyecto TANGIBLE [9], el cual se basa en la librería gráfica *OpenGL*. Por tanto, optamos por la utilización del leguaje de *shaders* a alto nivel *GLSL*, el cual, pese a no alcanzar la potencia gráfica de las últimas versiones de *DirectX*, presenta una mayor compatibilidad con sistemas no propietarios (por ejemplo basados en *Linux*).

Para que en el sistema se pudieran utilizar *shaders* se utilizó el framework *Ogre* debido a su amplio soporte y versatilidad en lo referente a técnicas avanzadas de gráficos. En primer lugar hay que poner tanto los programas del *shader* como los archivos que estos utilizan en una ruta visible desde la aplicación y hacer la llamada al shader en el archivo de materiales de la aplicación. *Ogre* tiene para cada escena un archivo .material en el que se define un material para cada objeto que se ha cargado en esa escena, en este archivo se utiliza el lenguaje de materiales que provee *Ogre*. En este archivo también se declaran los *shaders* que se quieren utilizar y se realizan las

llamadas a estos. Al hacer la llamada al *vertex shader* y al *pixel shader* se le pasan los parámetros y configuraciones necesarias.

El código de los *shaders* se pone en dos archivos independientes, el *vertex shader* en un archivo con la extensión .vert y el *píxel shader* en otro con la extensión .frag, aunque estas extensiones no eran necesarias se decidió hacerlo así ya que es una muy buena práctica de nomenclatura para diferenciar con facilidad los archivos que teníamos en la aplicación.

El *verter shader*, ejecutado previamente para cada vértice del objeto, realiza cálculos que posteriormente se utilizarán en el *píxel shader*. Éste, utilizando los datos obtenidos, realiza los cálculos necesarios para obtener el color adecuado de cada píxel. La Figura 2.1 describe el funcionamiento del *pipeline* de *GLSL+openGL*, mostrando en color naranja el paso por el *vertex shader* y el *píxel shader* (también llamado *fragment shader*).

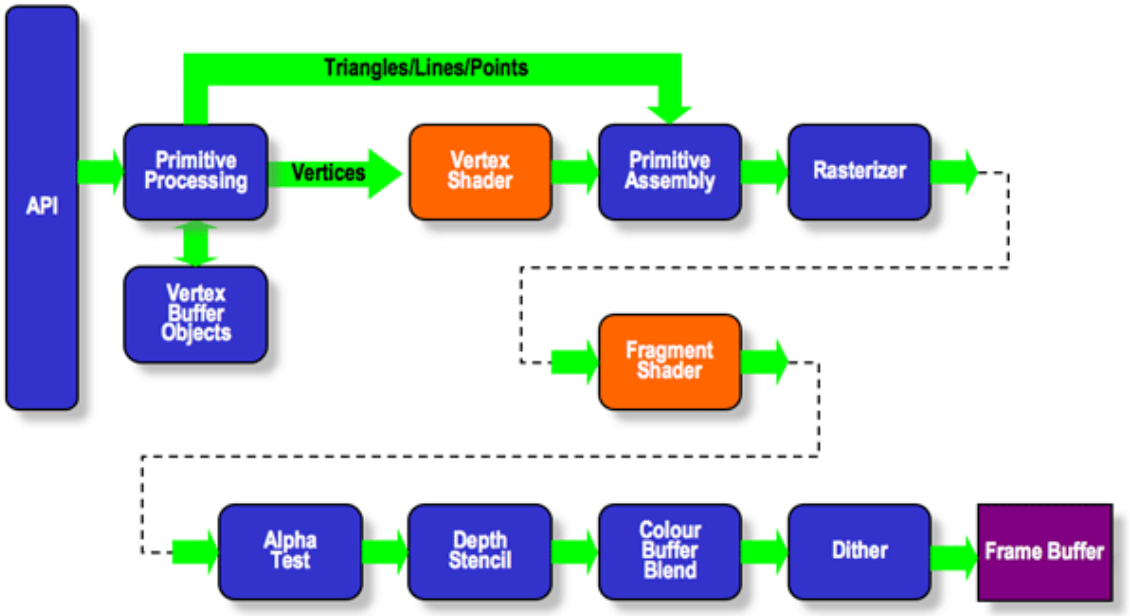

Figura 2.1 Pipeline de GLSL+OpenGL

Para demostrar el funcionamiento de todas estas características y comprobar su correcta integración con el resto de funcionalidades de la aplicación se implementó un *shader* de *parallax mapping* [10]. Esta es una novedosa técnica que se introdujo en el año 2001 que está basada en la técnica de *Normal mapping* pero introduciendo significativas mejoras. El objetivo de la técnica de *Parallax Mapping* es intentar dar la sensación de profundidad a las texturas sin utilizar mayor número de polígonos que es lo que provoca mayor tiempo de cálculo en el resultado final de la escena. Para ello lo que hace es desplazar las coordenadas de textura de un punto en función del ángulo existente entre el punto de vista y el valor del mapa de altura en dicho punto. Cuando el ángulo de vista es más abrupto, las coordenadas de textura se desplazan más, y generan mayor sensación de profundidad como se puede apreciar en la Figura 2.2.

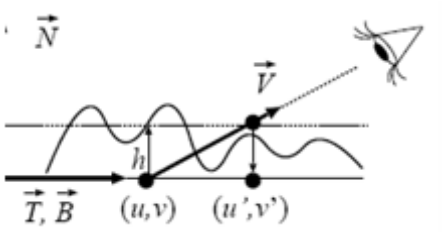

Figura 2.2 Técnica Parallax Mapping. Donde (u,v) son las coordenadas de textura originales, (u',v') son las coordenadas finales y h(u, v) es el desplazamiento

Para realizar estos cálculos se necesita tener pre-calculado el mapa de alturas para saber la profundidad de cada punto, el mapa de normales para saber en qué dirección apunta la normal en cada punto, y para el texturizado final y la iluminación hace falta la textura y el mapa de brillos respectivamente.

Observando la Figura 2.4 y la Figura 2.5 podemos ver la diferencia entre dos paredes de ladrillos en nuestra aplicación, una utilizando la técnica de *Parallax mapping* y la otra sin utilizarla. En ambos casos el objeto es el mismo, un objeto simple que consta solamente de dos caras planas (Figura 2.3). Sin embargo, esta técnica de texturizado consigue dar la sensación de volumen.

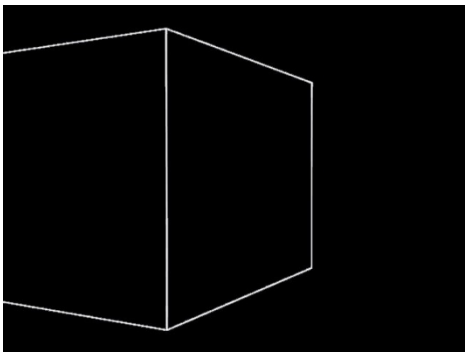

Figura 2.3 Poligonado del objeto

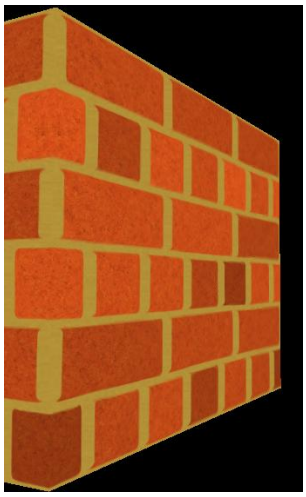

Figura 2.4 Pared de ladrillos sin parallax mapping

Figura 2.5 Pared de ladrillos con parallax mapping

2.3. Navegación

La aplicación dispone de un sistema para poder navegar por la escena, éste sistema consta de varias partes.

La primera funcionalidad del sistema consiste en el movimiento por la escena utilizando las teclas del teclado y el ratón, con las teclas W y A se puede avanzar y retroceder por la escena, con el movimiento del ratón manteniendo el botón derecho pulsado se puede mover arriba, abajo, derecha e izquierda por la escena. Por último con el movimiento del ratón manteniendo el botón izquierdo pulsado se puede modificar la dirección en la que se observa la escena. Es importante indicar que ésta dirección es siempre modificable por la posición indicada por el head tracking. De forma intuitiva podemos imaginar que modifica la posición en la que el usuario está encarado, mientras que el head tracking, captura su dirección en función de los movimientos de cabeza.

La segunda funcionalidad consiste en recoger la información de la posición de los sensores *Wiimote* que simulan los ojos del usuario, y con ésta realizar las modificaciones, simulando así el cambio de perspectiva de la escena respecto al cambio de posición del observador (ver Figura 2.6).

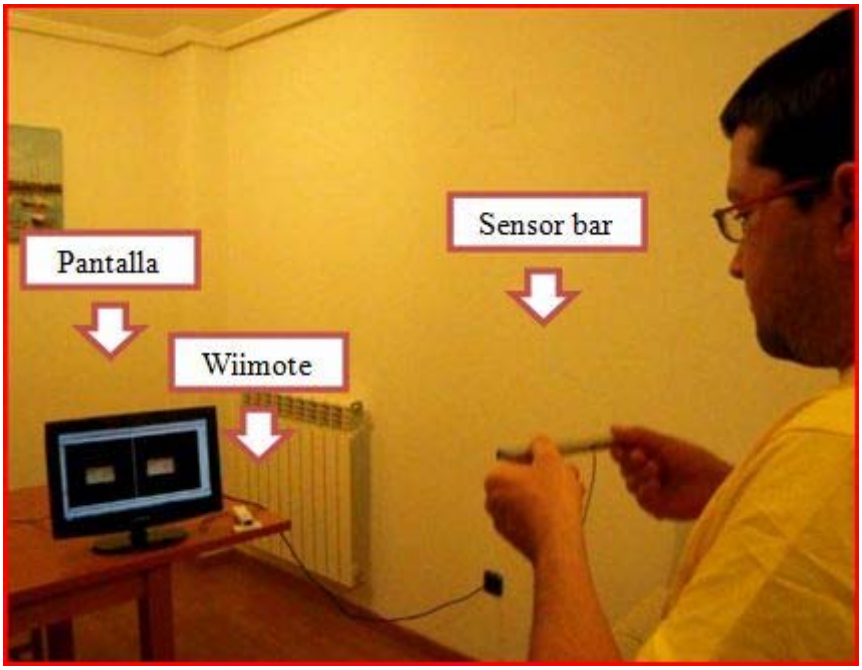

Figura 2.6 Utilización del sistema de posicionamiento.

2.4. Pruebas de carga

Uno de los requisitos de la aplicación era su correcto funcionamiento utilizando escenas con un número elevado de polígonos. Para comprobar el funcionamiento del sistema en estos casos se cargó en la aplicación una escena de este tipo y se realizaron

distintas pruebas sobre ella, comprobando las funcionalidades más importantes de la aplicación.

La escena utilizada para estas pruebas tenía 1.693.637 polígonos y correspondía a un modelo de arqueología virtual de un foro romano completo (ver Figura 2.7). Al arrancar la aplicación con esta escena se observó que el proceso era más lento, ya que el arranque llegaba a tardar incluso minutos, pero una vez arrancada la aplicación todas las funcionalidades del sistema funcionaban correctamente. Aunque, en los casos en los que la cámara enfocaba la parte central de la escena, donde estaba la mayor concentración de polígonos, existía cierto retardo en algunas operaciones. Sin embargo éste no impedía el uso normal de la aplicación.

Figura 2.7 Foro romano en la aplicación (Las dos imágenes que conforman la imagen en tres dimensiones)

Estas pruebas de carga fueron realizadas con un ordenador portátil convencional que disponía de un procesador Intel(R) Core(TM)2 Duo CPU T7300 @ 2.00GHz, 2GB de memora RAM y con una tarjeta ATI Radeon Xpress 1200 Series con una memoria de 256MB. Éste es hoy en día un equipo de perfil bajo con lo que en la mayoría de equipos actuales el rendimiento de la aplicación debería ser superior al obtenido en esta prueba.

2.5. XML de configuración

Para poder configurar las propiedades de la aplicación, ésta dispone de un fichero .xml en el cual el usuario puede inicializar el valor de sus principales parámetros, tal y como se describe a continuación.

El primer valor que se puede configurar es el la ruta de la escena a cargar, es decir, la ruta del archivo .osm que contiene la escena. El segundo es la posición de la cámara estereoscópica en la escena y el tercero es el punto hacia el que mira dicha cámara. El cuarto y quinto son la distancia entre los ojos y el grado de convergencia respectivamente, son los parámetros referentes a la estereoscopía. Los siguientes son los que configuran las luces de la escena, el sistema puede tener varias luces y para cada una de ellas se indica su posición en la escena, su tipo (*Spotlight*, *Point*, *Directional*), el color especular y el color difuso en RGB (Red Green Blue). Por último se puede configurar si se desea activar o no la funcionalidad de posicionamiento del usuario mediante el *Wiimote*. En la Figura 2.8 se aprecia con más claridad la estructura del archivo de configuración XML.

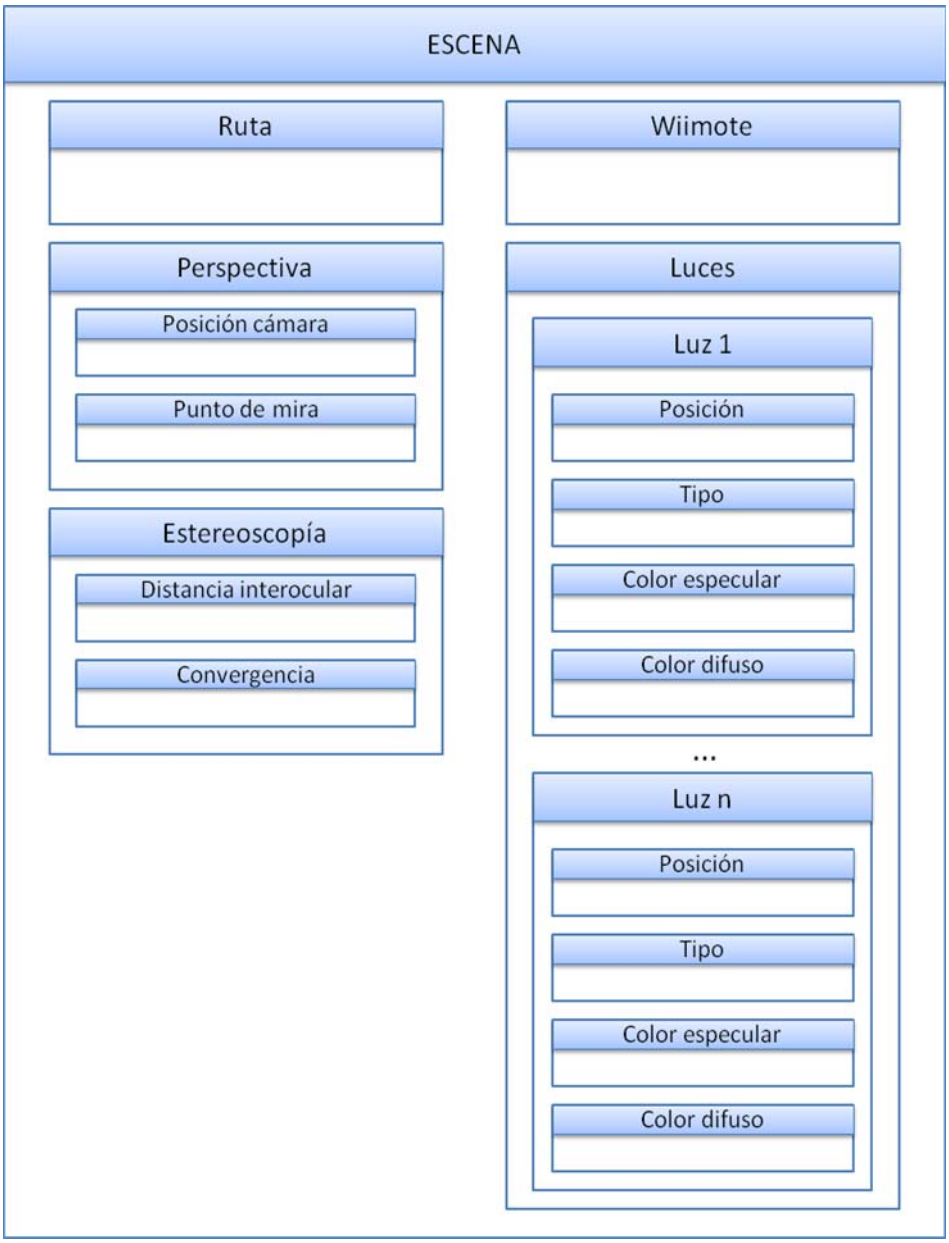

Figura 2.8 Estructura del archivo de configuración XML

Capítulo 3. Generación de estereoscopía

3.1. ¿Qué es?

Antes de comenzar la implementación del sistema estereoscópico se realizó un estudio previo con el fin de averiguar en qué consistía la estereoscopía y cuál era la forma adecuada de implementarla en nuestro sistema.

Estereoscopía [11] es cualquier técnica capaz de recoger información visual tridimensional de una escena o crear la ilusión de profundidad en una imagen. Literalmente, estereoscopía significa ver con dos ojos. La técnica utilizada consiste en obtener dos imágenes distintas, una para cada ojo, de la escena. Estas están tomadas a 6,5 cm de distancia una de la otra que es la distancia entre los ojos media en un ser humano, simulando así que cada ojo ve la perspectiva que vería en la realidad. La Figura 3.1 representa gráficamente esta idea.

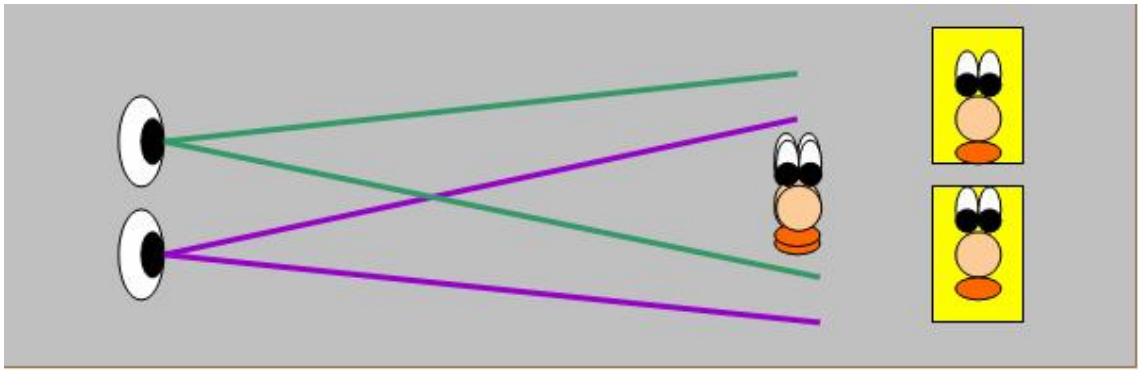

Figura 3.1 Representación gráfica de estereoscopía

Todo esto es posible gracias a que el ser humano tiene dos ojos situados a unos 6.5 cm y estos recogen dos imágenes ligeramente distintas uno del otro de la misma escena. Son estas pequeñas diferencias las que utiliza el cerebro para calcular la distancia de los objetos, esta técnica es conocida como paralaje. Este cálculo de distancias es el que da al cerebro la tercera dimensión y por tanto la sensación de profundidad. Por lo tanto cuando recogemos dos imágenes ligeramente distintas una de otra tomadas a una distancia de 6.5 cm una de otra y le mostramos al usuario una en cada ojo, el cerebro puede reconstruir la distancia y por tanto tener la sensación de tridimensionalidad.

3.2. Profundidad de campo

La profundidad de campo es la zona visualizada por la aplicación en la que la imagen resultante es nítida. Por tanto, los objetos que se encuentren en la zona de la imagen correspondiente a la profundidad de campo se deben ver nítidos. En nuestro caso, al estar trabajando con un sistema de visión estereoscópica, en la profundidad de campo influyen, además de los factores típicos como el enfoque de la cámara, otros como la convergencia.

3.2.1. Convergencia

Se llama punto de convergencia al punto en el que los ejes de las dos cámaras que conforman la cámara estereoscópica convergen para crear la imagen resultante en 3 dimensiones, es decir, el punto al que las dos cámaras apuntan. Podemos verlo con más claridad en la Figura 3.2.

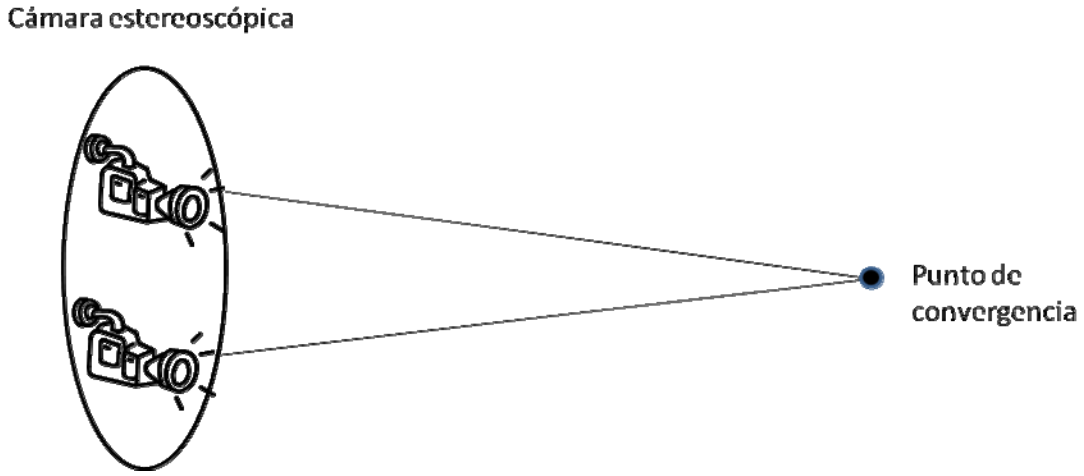

Figura 3.2 Punto de convergencia

En la aplicación desarrollada, para que el usuario pudiera adaptar la convergencia del sistema estereoscópico a sus necesidades, se ha implementado un sistema que permite variar el grado de convergencia de la cámara estereoscópica en un punto. De esta forma, se puede modificar desde totalmente convergente en el punto deseado hasta totalmente paralelos los ejes de las dos cámaras, como se puede apreciar en la Figura 3.3.

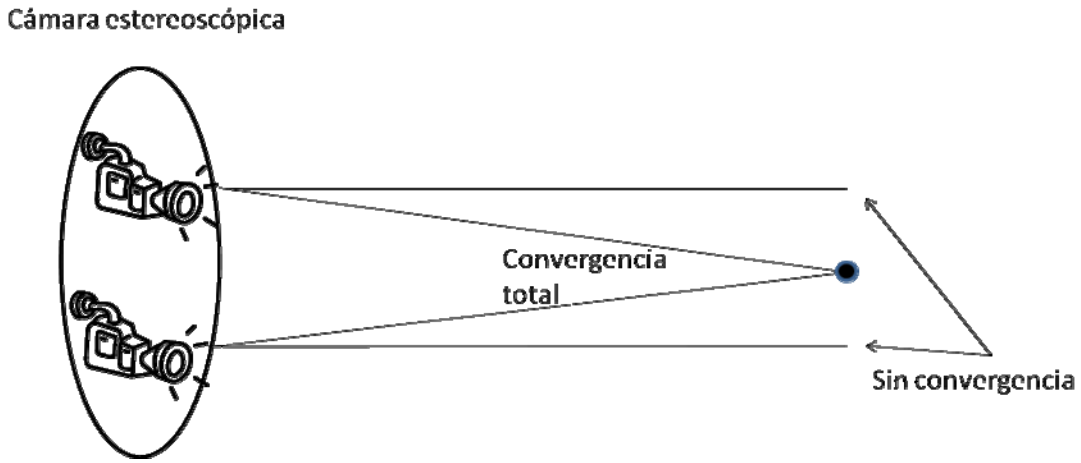

Figura 3.3 Convergencia total

3.2.2. Sensación

Los elementos mencionados anteriormente influyen de manera directa en la sensación que percibe el usuario de la aplicación.

Por ejemplo, si el punto en el que centra la atención del usuario no se encuentra dentro de la profundidad de campo, el usuario verá el punto borroso y esto puede provocar en él una sensación de mareo. Esto es debido a que el objeto que está mirando está desenfocado y sin embargo el objeto al que no está mirando está enfocado.

Con la convergencia estereoscópica pueden ocurrir problemas parecidos ya que si no hay convergencia de la cámara estereoscópica en el punto al que el usuario está mirando este no va a apreciar el efecto 3D, podemos apreciar mejor este caso en la Figura 3.4.

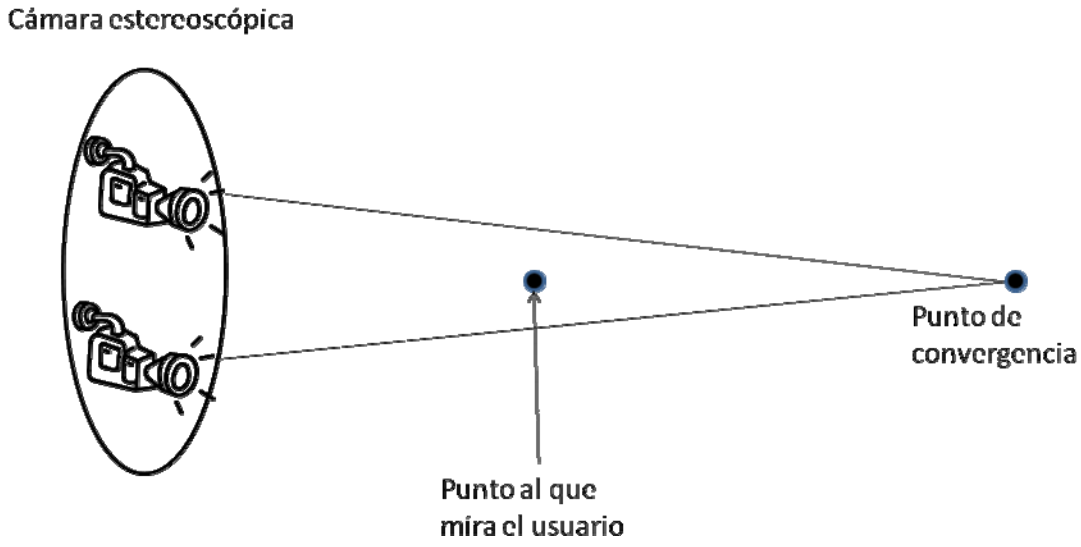

Figura 3.4 Problema con la estereoscopía si no hay convergencia en el punto al que el usuario mira

Si el usuario está mirando a un objeto que está más lejos que el punto de convergencia (Figura 3.5) existirá el mismo problema, y además puede producirse una sensación extraña, ya que al estar detrás del punto de convergencia puede producirse el mismo efecto que el utilizado para producir la sensación de 3 dimensiones pero al revés. Se ven los objetos de dos formas distintas, una con cada cámara, pero no encaja con lo que vería cada cámara en una situación adecuada de 3D, por lo que el cerebro no es capaz de interpretarlo y puede producir mareos.

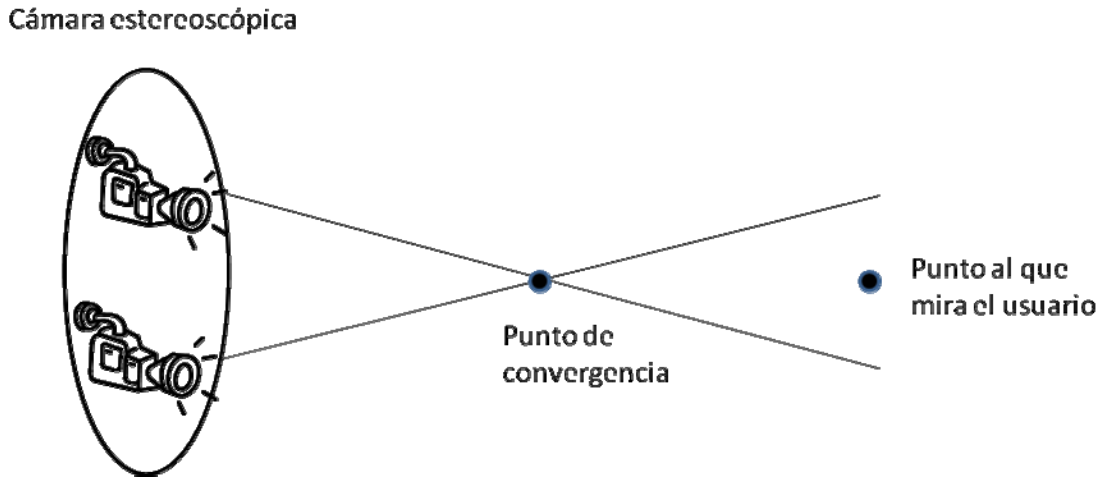

Figura 3.5 Problema con la estereoscopía cuando el punto al que mira el usuario está detrás del punto de convergencia

Por el contrario, si la convergencia es muy grande porque el objeto al que se está enfocando está muy cerca (Figura 3.6) también se puede provocar al usuario una sensación de mareo, de la misma forma que ocurre en la realidad.

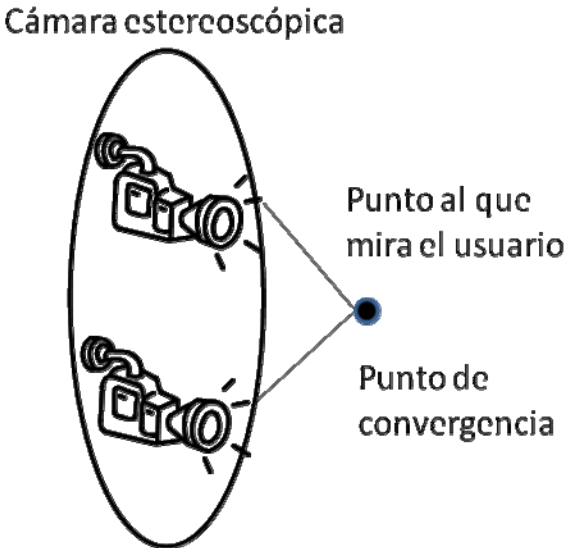

Figura 3.6 Objeto enfocado demasiado cerca

Además, como la convergencia influye en la profundidad de campo, ésta puede afectar a la percepción que el usuario tiene sobre la escala de la escena. Un ejemplo de que la profundidad de campo influye en la percepción de la escala es el efecto *Tilt Shift*

[20] (Figura 3.7), que consiste en reducir la profundidad de campo, desenfocando las partes más lejanas de la imagen y provocando un efecto de miniatura.

Figura 3.7 Efecto Tilt Shift

3.2.3. Configuración en XML

Como para cada escena los parámetros más adecuados en lo referente a la estereoscopía pueden variar sustancialmente, estos parámetros se introdujeron en el archivo de configuración de la aplicación, permitiendo así que el sistema se adapte en cada caso a las necesidades de la escena.

El archivo .xml permite configurar el grado de convergencia, el punto al que enfoca la cámara y la distancia entre los ojos.

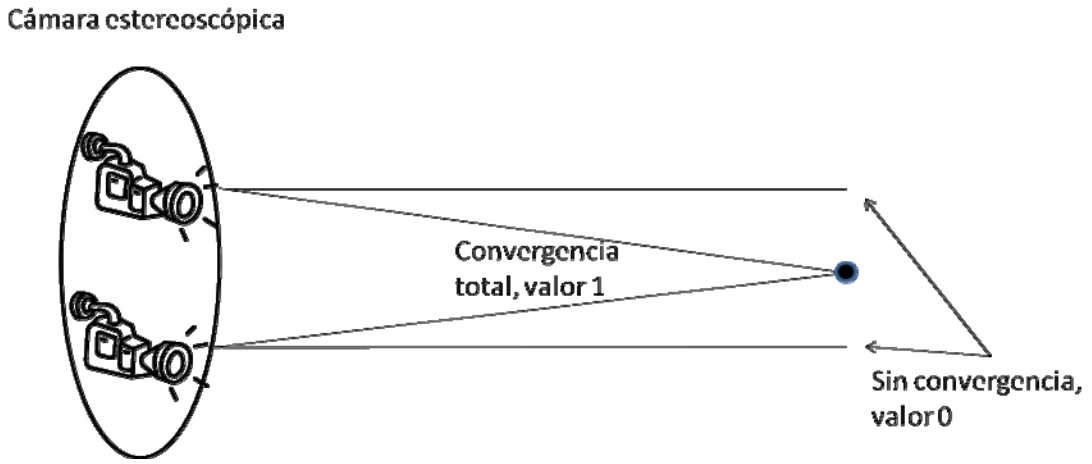

Figura 3.8 Posibles valores que puede tomar el grado de convergencia

El grado de convergencia puede tomar un valor entre cero y uno, cero significaría que no existe convergencia por lo que las dos cámaras que conforman la cámara estereoscópica estarían totalmente paralelas. Por el contrario, el valor uno significaría que existe una convergencia total al punto al que está apuntando la cámara

estereoscópica. Para los valores que están entre el cero y el uno la convergencia existiría pero no sería total, conforme el valor aumente aumenta el grado de convergencia (Figura 3.8).

El punto al que apunta la cámara estereoscópica, junto con el grado de convergencia, es el que determina el punto que va a tomar la aplicación como referencia para calcular la dirección a la que debe mirar cada una de las cámaras que conforman la cámara estereoscópica.

Por otro lado, la distancia entre los ojos es la distancia que separa las dos cámaras que conforman la cámara estereoscópica. Variar este parámetro puede ser muy útil ya que aumenta o disminuye la sensación de profundidad en la escena y con ello la de tridimensionalidad. El valor en este fichero de configuración debe de ser introducido en centímetros.

Capítulo 4. Interacción

4.1. Navegación

La aplicación dispone de un sistema de navegación que permite al usuario moverse por la escena.

Dicho sistema dispone de varias herramientas que son las que proporcionan la interacción entre el usuario y la escena. La primera de ellas es el *HeadTraking*, mediante el cual se recoge la posición del usuario y se modifica la perspectiva de la escena con respecto a ésta. La segunda herramienta es una librería que permite el movimiento del usuario por la escena, ésta dispone de funciones para mover la cámara arriba, abajo, derecha o izquierda; funciones para avanzar, retroceder y para modificar la dirección en la que mira el usuario.

El sistema de navegación desarrollado está preparado para integrarse con distintos controles, por ejemplo con la biblioteca de gestos que está incluida en TANGIBLE o con el ratón y el teclado.

4.2. HeadTraking

El sistema está preparado para recoger la información de la posición de los sensores *Wiimote* que simulan los ojos del usuario. A partir de dicha posición, se realizan las modificaciones necesarias para simular el cambio de perspectiva de la escena respecto al cambio de posición del observador.

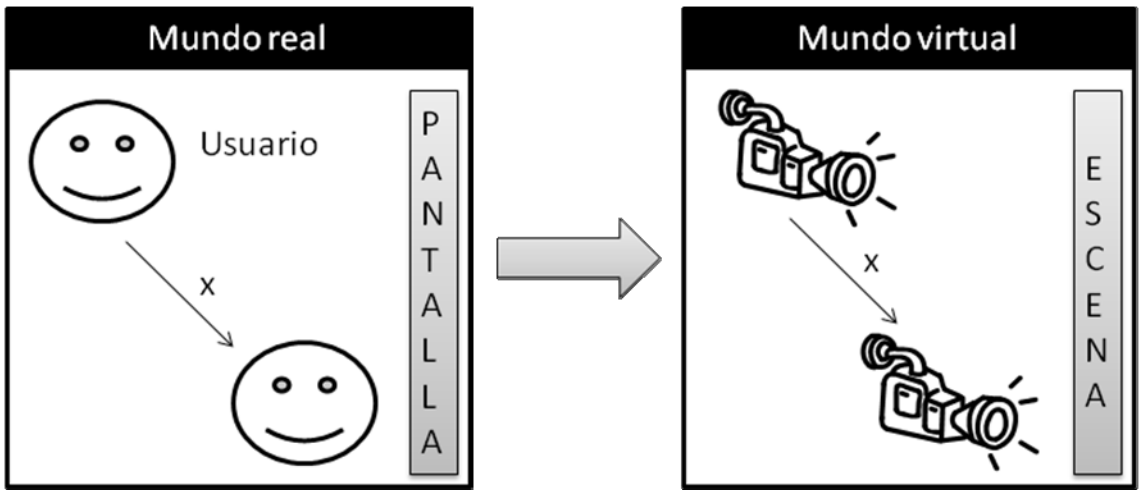

Figura 4.1 Uso de HeadTracking en el sistema de navegación

Para implementar este sistema se ha utilizado la librería *WiimoteLib* que ha servido para obtener la información tridimensional de la posición del usuario. Una vez se tiene esta información, se compara con la posición anterior obtenida para saber cuál ha sido el desplazamiento del usuario. Este desplazamiento se lo aplicamos a la cámara de la escena, simulando de esta forma el movimiento del usuario en la realidad. Esta idea puede verse gráficamente en la Figura 4.1.

Para la conexión entre el *Wiimote* y la aplicación se ha utilizado el sistema *bluetooth*, que permite conectar un *Wiimote* con un PC. En la Figura 4.2 se puede ver un ejemplo real de cómo se traslada el movimiento del mundo real al mundo virtual.

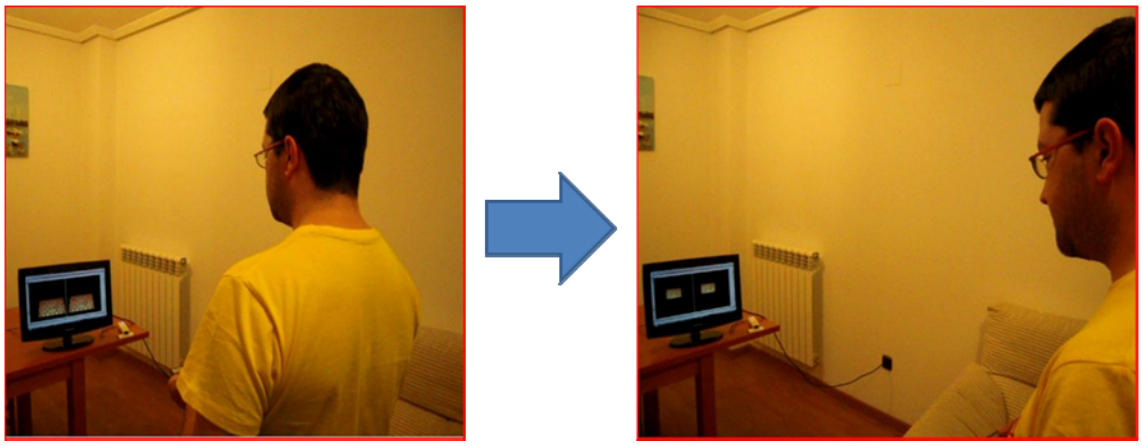

Figura 4.2 Ejemplo de desplazamiento hacia atrás de la Sensor bar en el mundo real que provoca el alejamiento del objeto en el mundo virtual

Uno de los objetivos del proyecto era construir un entorno inmersivo. La inmersión consiste en hacer sentir al usuario que forma parte del mundo virtual que se ha creado.

Esta es la herramienta que hace que la inmersividad del sistema sea elevada, ya que al modificarse la perspectiva de la escena con respecto a la posición del usuario, éste tiene la sensación de que forma parte de la propia escena. Esto es debido a que la respuesta del sistema ante los movimientos es muy similar al comportamiento real del ojo humano cuando nos movemos, la perspectiva percibida se modifica al cambiar la posición.

4.3. Gestos para navegar

El sistema dispone de una librería que dota a la aplicación de las funciones necesarias para permitir el movimiento del usuario por la escena.

En concreto, dispone de las funciones de mover la cámara hacia arriba, abajo, derecha o izquierda. Así como de avanzar, retroceder o de modificar la dirección en la que mira el usuario.

El sistema está preparado para integrarse con distintos controles, como por ejemplo con la biblioteca de gestos [12] que está incluida en TANGIBLE. Sin embargo, de momento los controles utilizados han sido el ratón y el teclado, con los cuales se ha comprobado el correcto funcionamiento de la navegación y se han hecho las pruebas pertinentes.

La biblioteca de gestos incluida en TANGIBLE reconoce una gran cantidad de ellos, y estos pueden ser utilizados para asociarlos con movimientos del usuario, consiguiendo así que el usuario pueda navegar por la escena realizando cualquiera de estos gestos con sus manos. Los gestos simples reconocidos son los que se aprecian en la Figura 4.3, además reconoce decenas de gestos compuestos. Estos gestos tienen una asociación directa con los distintos movimientos dentro de la escena que permite el sistema de navegación.

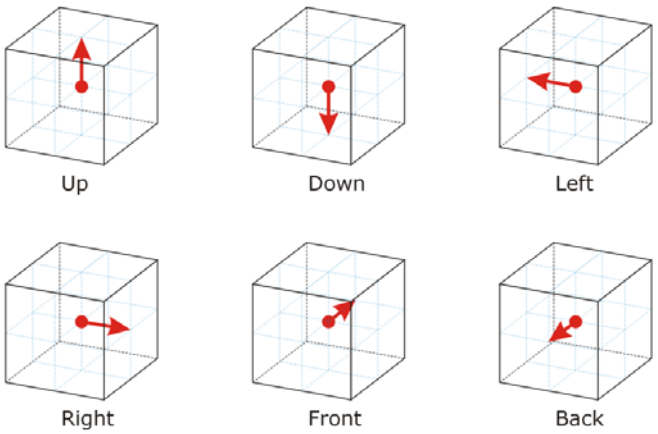

Figura 4.3 Gestos simples reconocibles

Esta biblioteca es capaz de calcular la posición espacial y reconocer gestos basados en patrones de uno o varios emisores de infrarrojos captándolos con las cámaras que están integradas en los mandos de la consola *Wii*. Los mandos comunican los datos al PC a través del protocolo B*luetooth.* Además, proporciona funciones que, utilizadas en la aplicación, nos permiten saber el gesto que el usuario ha realizado. Una vez que se dispone de esta información nosotros podemos a asociarla a uno de los movimientos de los que dispone el sistema de navegación del usuario. De esta forma, cuando el usuario realice un gesto concreto la aplicación realizará el movimiento de navegación asociado a este.

Capítulo 5. Descripción de la aplicación final

5.1. Esquema de comunicación

En el diagrama de la Figura 5.1 se describe la estructura de la aplicación. Los rectángulos corresponden a los distintos componentes del sistema, tanto físicos como de software. Las elipses representan las comunicaciones entre unos componentes y otros.

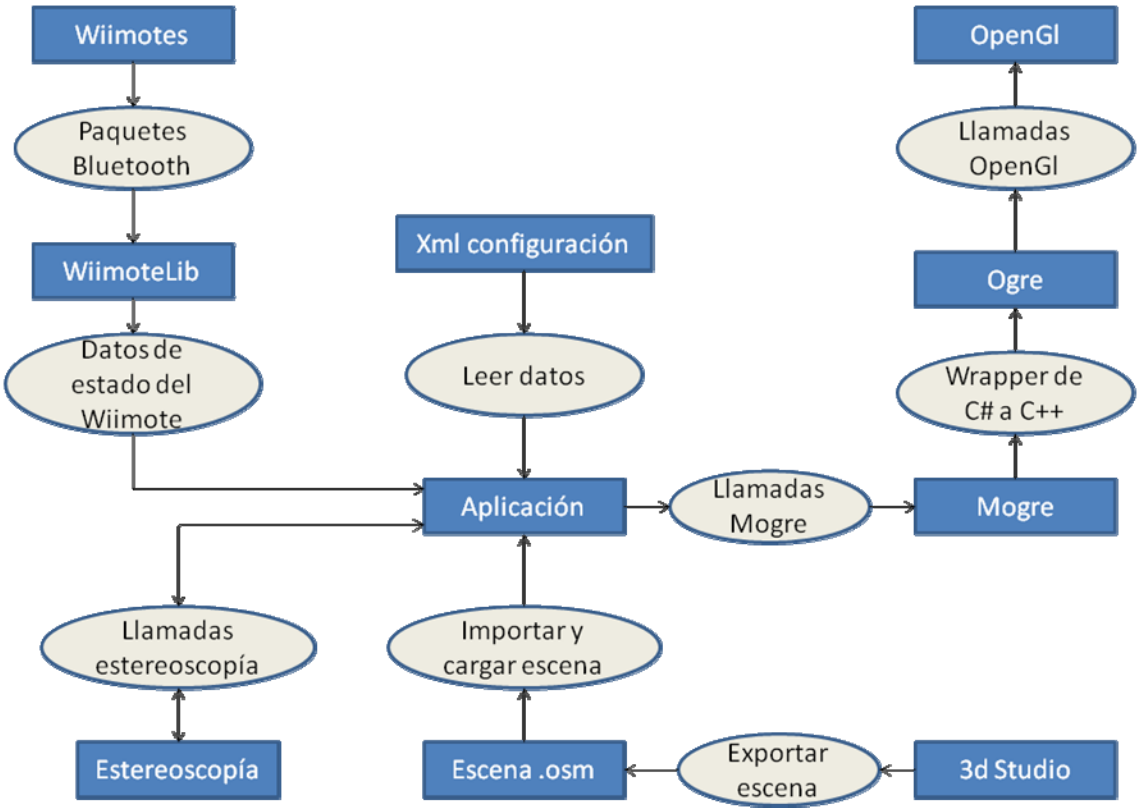

Figura 5.1 Esquema de comunicación de la aplicación

5.2. Sistema Físico

Los componentes físicos utilizados y en los que se basan tanto el sistema de interacción, como el sistema de estereoscopía de la aplicación son los siguientes: *Wiimote*, Antena *Bluetooth USB, Wii Sensor bar*, PC con sistema operativo Windows y con una tarjeta gráfica que permita estereoscopía y por último proyector estereoscópico (Figura 5.2).

Figura 5.2 Componentes físicos utilizados

5.3. Descripción de las clases de la aplicación

A continuación, se muestran los diagramas de clases de las principales clases de la aplicación desarrollada, así como una breve explicación acerca de cada una de ellas.

5.3.1. Clase Stereoscopycam

La clase Stereoscopycam representa una cámara estereoscópica y dispone de los métodos y atributos necesarios para abstraerse de su implementación interna, permitiendo instanciar y utilizar cámaras estereoscópicas. En la Figura 5.3 se muestra su diagrama de clase.

5.3.2. Clase OSMLoader

La clase OSMLoader proporciona los métodos y atributos necesarios para cargar una escena en la aplicación desde un fichero en formato .osm. En la Figura 5.4 puede observarse su diagrama de clase.

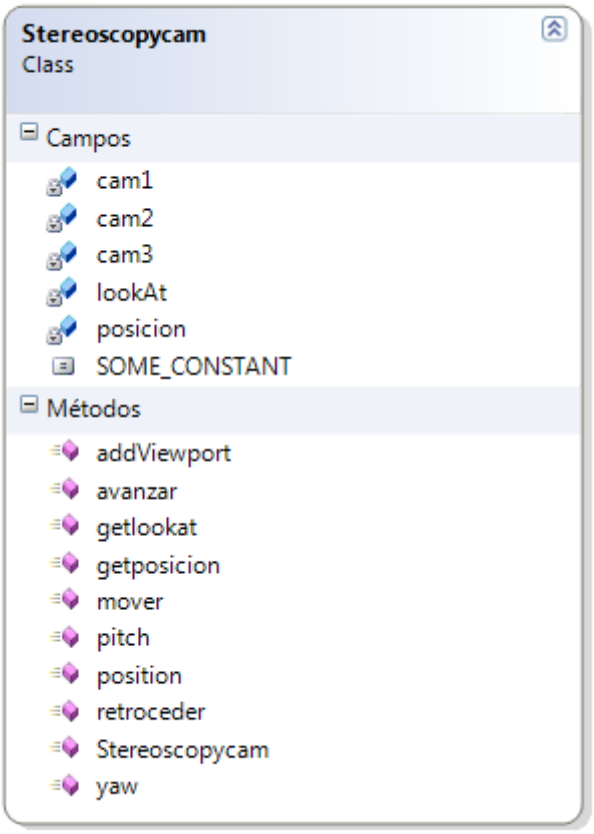

Figura 5.3 Diagrama de clase de Stereoscopycam

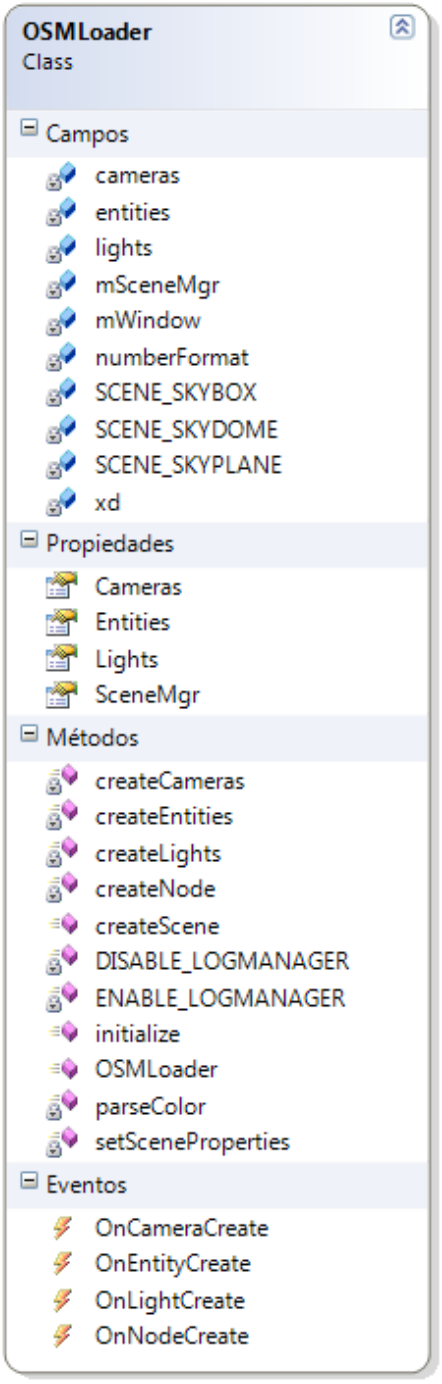

Figura 5.4 Diagrama de clase de OSMLoader

5.3.3. Clase OgreForm

La clase OgreForm se utiliza para crear el diálogo de la aplicación, crear las pantallas de renderizado y poder interactuar con el usuario. En el campo cam se guarda la cámara estereoscópica (clase Stereoscopycam). Al inicializar la clase OgreForm, cuando se arranca la aplicación, se instancia un objeto de la clase OSMLoader que se utiliza para cargar la escena configurada. En la Figura 5.5 se muestra el diagrama de clase de OgreForm.

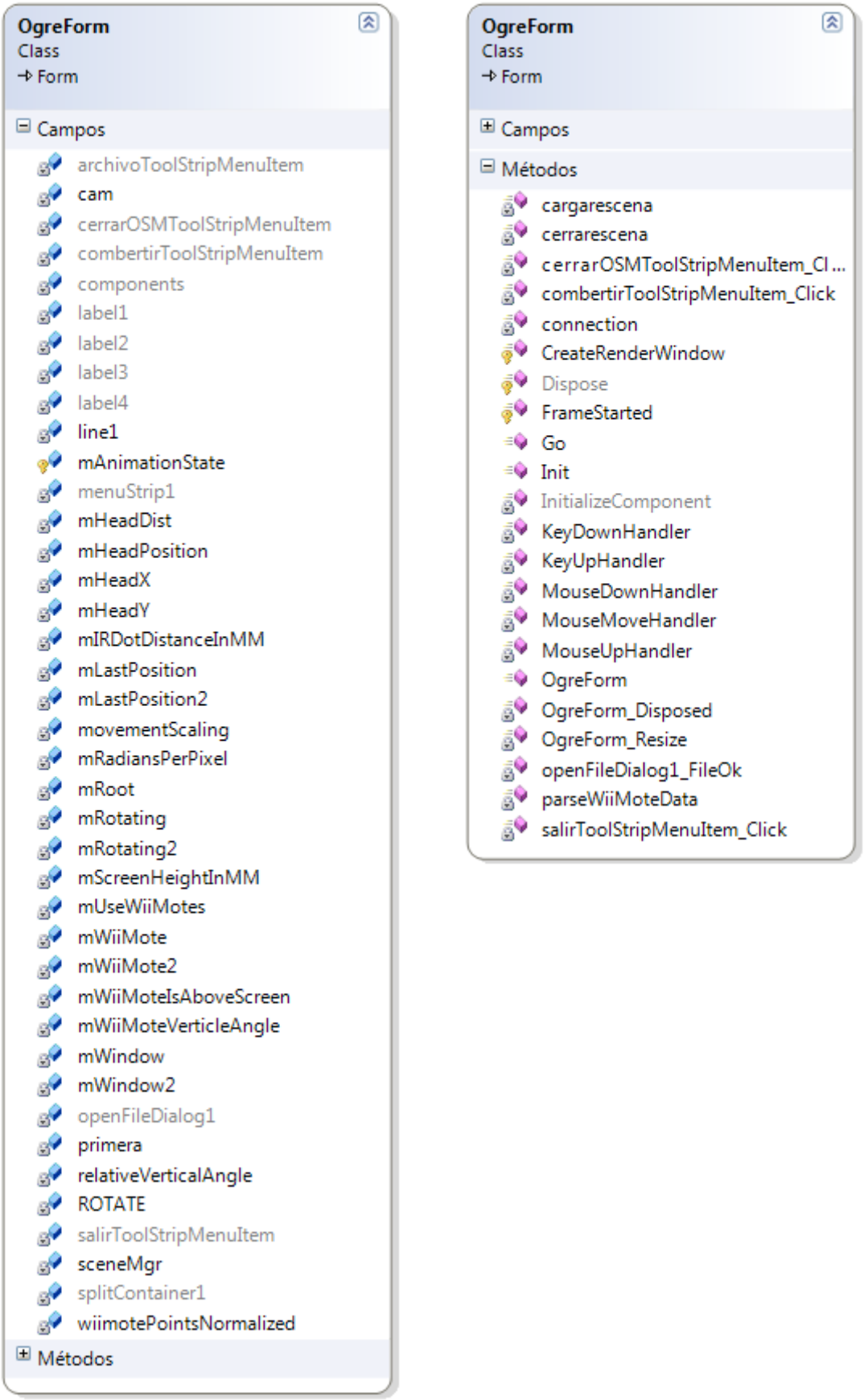

Figura 5.5 Diagrama de clase de OgreForm

Capítulo 6. Resultados y conclusiones

6.1. Conclusiones personales

6.1.1. Beneficios

Durante el desarrollo de este proyecto he investigado, desarrollado y documentado cada uno de los componentes necesarios. Para ello, se ha seguido una metodología profesional que me ha servido como aprendizaje para el futuro.

El desarrollo de este proyecto con resultados prácticos claramente visibles, a diferencia de programas realizados para la propia formación, ha sido motivador para el futuro.

Además, que el proyecto se haya llevado a cabo utilizando el lenguaje de programación C# y el motor de render *Ogre* ha supuesto el aprendizaje de herramientas modernas, distintas a las vistas anteriormente en el transcurso de la carrera. Esto puede serme de gran ayuda para la utilización de las mismas en el futuro a nivel profesional.

6.1.2. Dificultades

La parte correspondiente al estudio de los *shader* y el trabajo con los mismos ha supuesto muchas horas de estudio e implementación, especialmente la parte de comunicación entre la aplicación, el *shader* y las diferentes formas existentes para pre-calcular los datos que el *shader* necesitaba (mapas de normales, de brillo, de altura, de textura).

Otra de las partes del proyecto que también ha supuesto gran cantidad de horas de trabajo ha sido la investigación en el campo de la estereoscopía y el posterior desarrollo de la clase que proporciona el sistema estereoscópico a la aplicación. Se requería de un trabajo de investigación acerca de cómo crear un sistema estereoscópico realista. Esta investigación nos llevó a un sistema que entraña una cantidad de cálculos matemáticos de cierta complejidad para conseguir el correcto posicionamiento de las dos cámaras que conforman la cámara estereoscópica.

6.1.3. Enfoque

El enfoque inicial fue bastante acertado. Consistía en realizar una investigación y estudio previo de las opciones disponibles para cada una de las partes del proyecto, analizándolas y optando por la opción más ventajosa en cada caso. Para tomar las

decisiones se tendría también en cuenta el proyecto de manera global, de forma que todas las piezas encajaran correctamente.

Sin embargo, hubo aspectos en los cuales el trabajo de investigación previo no sirvió para determinar la solución más adecuada, y fue en la fase de desarrollo cuando se detectaron algunos problemas, implicando volver a la fase de estudio y realizar otra iteración. Por ejemplo, tras estudiar los diferentes exportadores de objetos, se decidió utilizar *OgreMax*, pero al empezar a utilizarlo se presentaron algunos errores y se comprobó que no era el más adecuado para el desarrollo del proyecto.

6.2. Resultados y aplicaciones

Los objetivos iniciales se han cumplido en su totalidad. Al finalizar el proyecto, surgen ahora nuevas posibilidades de desarrollo y aplicación. Algunas de las más importantes son las que se describen en este apartado.

6.2.1. Virtual Heritage

Virtual Heritage [14] o Arqueología Virtual es un concepto que aparece a principios de la década de los 90.

Según Paul Reilly, que fue el primero en proponer este concepto, se define como “el conjunto de técnicas informáticas que permiten la visualización 3D de la representación virtual y realista de los objetos y edificios antiguos, cuyos restos han desaparecido o están en un estado de preservación tan deficiente que hacen imposible su observación o muy difícil su interpretación”.

La aplicación desarrollada en este proyecto, por sus características de visualización y sus capacidades de interacción, permite la inmersión del usuario en la escena que se está visualizando, haciéndole sentir que forma parte de ella. Por tanto, se perfila como un sistema muy apropiado para introducirse en reconstrucciones de ruinas arqueológicas, permitiendo al usuario observarlas, recorrerlas y comprenderlas.

Figura 6.1 Reconstrucción del teatro romano de Bilbilis

En la Figura 6.1 puede observarse la reconstrucción del teatro romano de Bilbilis [16], que es un claro ejemplo de arqueología virtual. El presente proyecto permite la reproducción a este nivel de calidad visual en tiempo real.

6.2.2. Biomedicina

Una de las aplicaciones de los entornos inmersivos, como el desarrollado en este proyecto, es el campo de la biomedicina. En la Figura 6.2 se observa un ejemplo de un entorno inmersivo utilizado en este campo [15].

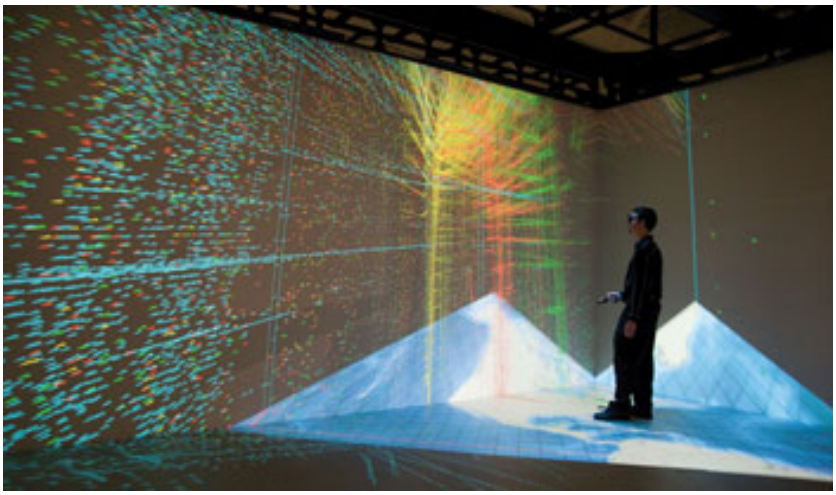

Figura 6.2 Entorno inmersivo utilizado en biomedicina

En el caso concreto de este proyecto, podría utilizarse para visualizar escenas relacionadas con este campo y, además, permitir al usuario desplazarse a través de las mismas.

Por un lado, podrían cargarse escenas que representarán las distintas partes del cuerpo humano, de tal forma que los médicos visualizaran ciertos casos y pudieran practicar antes de enfrentarse a ellos en el mundo real. Por otro lado, también se podría utilizar para cargar, analizar y comprender escenas representando elementos que, por su tamaño, no son visibles al ojo humano, como por ejemplo el ADN.

En resumen, la aplicación podría ser especialmente útil en el campo de la formación médica, permitiendo a los médicos o estudiantes adentrarse en el cuerpo humano como si tuvieran un tamaño microscópico.

6.2.3. Realidad Aumentada

Se define como Realidad Aumentada a la visión del mundo real, en la que sus elementos se combinan con elementos virtuales para crear una realidad mixta (parte real y parte virtual), todo ello en tiempo real. En la Figura 6.3 puede verse un ejemplo de realidad aumentada en el que la mano y el cromo serían la parte real y el jugador de béisbol sería la parte virtual [17].

Figura 6.3 Ejemplo de realidad aumentada

En el caso de este proyecto, se podría proyectar la escena cargada en la aplicación sobre objetos reales, dando así la sensación de que se añade la realidad virtual a una escena del todo real. Esto podría ser utilizado en el mundo de la arqueología, ya que se podrían recrear partes de ruinas que estuvieran en mal estado de conservación o deterioradas sobre la parte real existente. De esta forma, el visitante de las ruinas tendría la sensación de que está visualizándolas tal y como se encontraban antes de su deterioro.

Por un lado, la estereoscopía nos permitiría visualizar la parte virtual en 3 dimensiones y, por otro lado, el *HeadTracking* permitiría que la parte virtual se visualice con la misma perspectiva que la parte real. La suma de estas dos características permitiría que las dos partes (real y virtual) quedaran totalmente acopladas.

6.2.4. Realidad virtual

El concepto de realidad virtual consiste en producir virtualmente al usuario una apariencia de realidad generándole la sensación de estar en ella (En la Figura 6.4 se observa un ejemplo [18]).

El sistema desarrollado se puede utilizar como sistema de realidad virtual, ya que, gracias a características como el posicionamiento del usuario, el sistema de estereoscopía (que aporta sensación de profundidad) o la alta calidad gráfica soportada en las escenas cargadas, produce en el usuario la sensación de estar presente en un entorno real.

Figura 6.4 Ejemplo de realidad virtual